

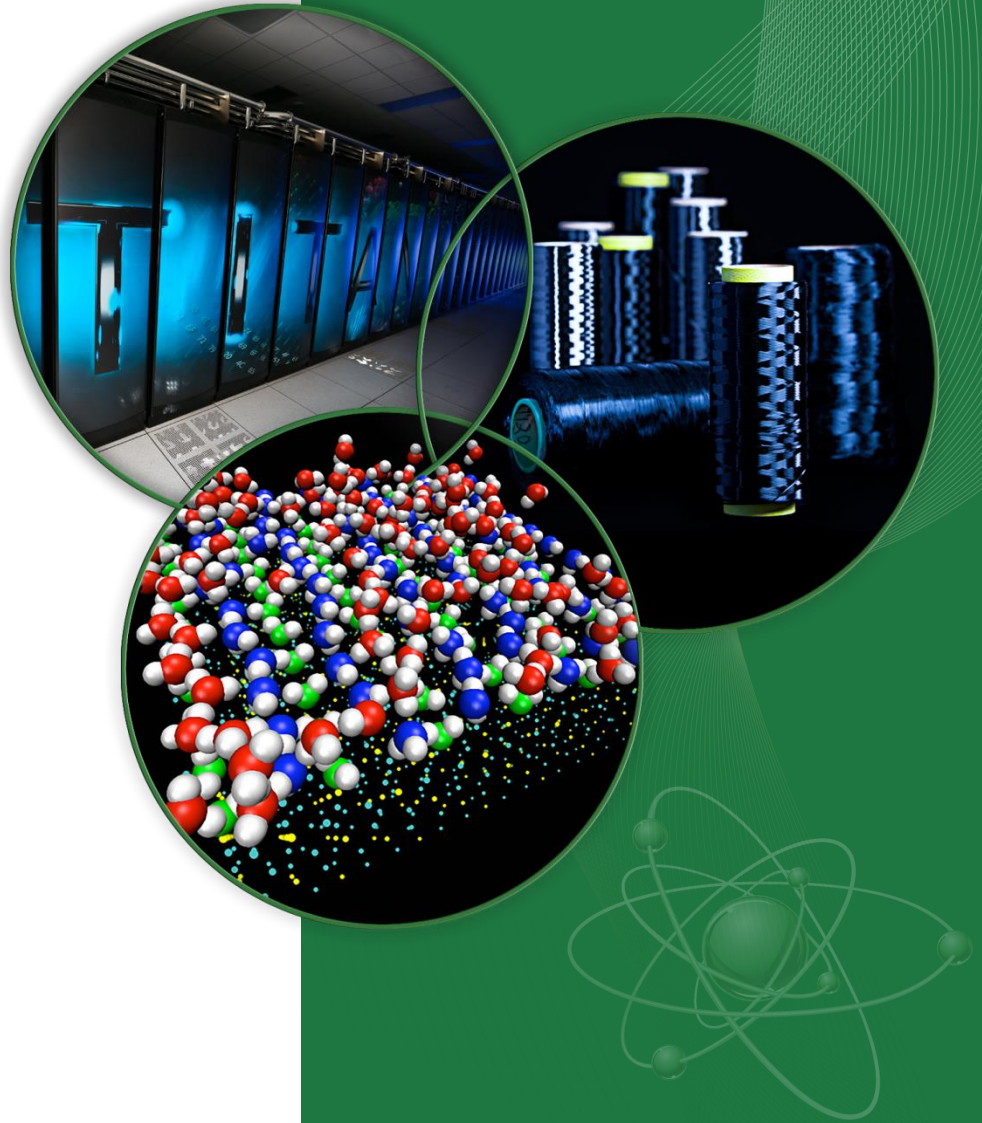
# EPICS State Notation Language (SNL), “Sequencer”

Kay Kasemir,  
SNS/ORNL

Many slides from  
Andrew Johnson,  
APS/ANL

June 2014

ORNL is managed by UT-Battelle  
for the US Department of Energy

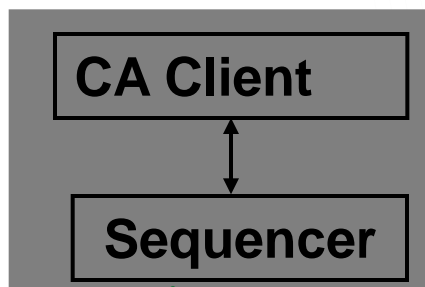
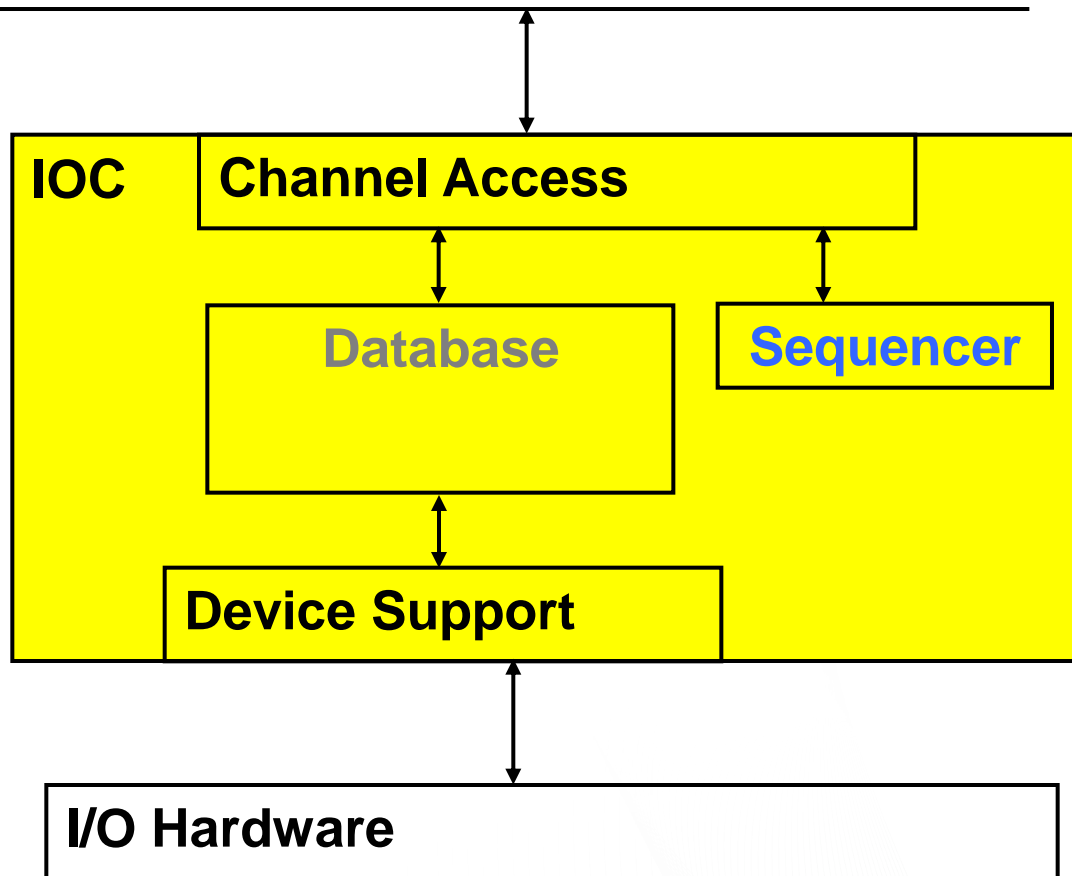


# IOC

- Database:  
Data Flow,  
mostly periodic  
processing
- Sequencer:  
State machine,  
mostly  
on-demand

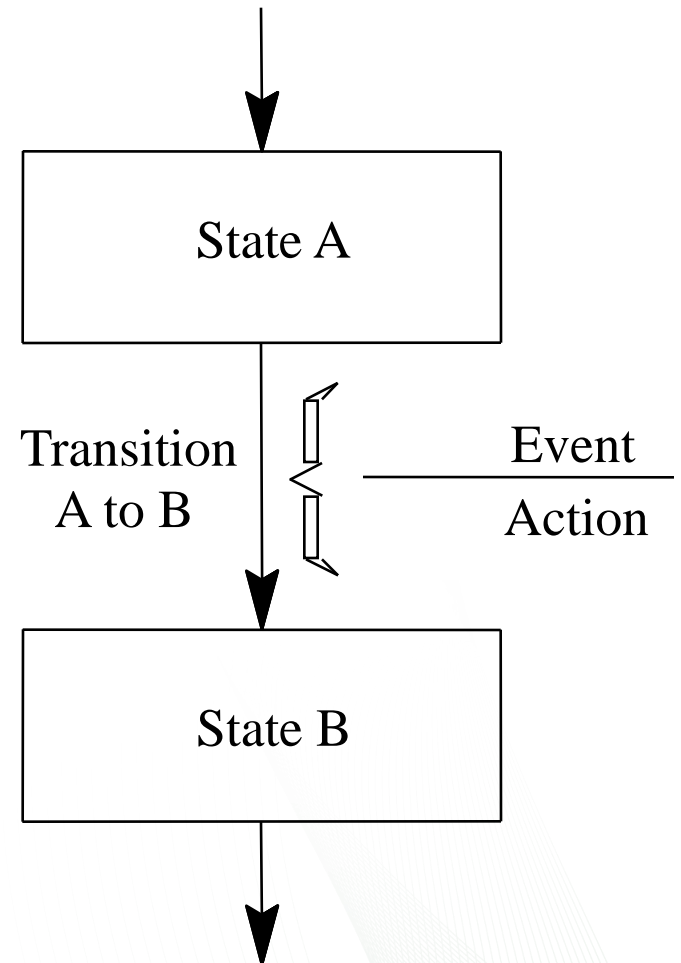
Optional:  
Sequencer runs as  
standalone CA-Client

LAN

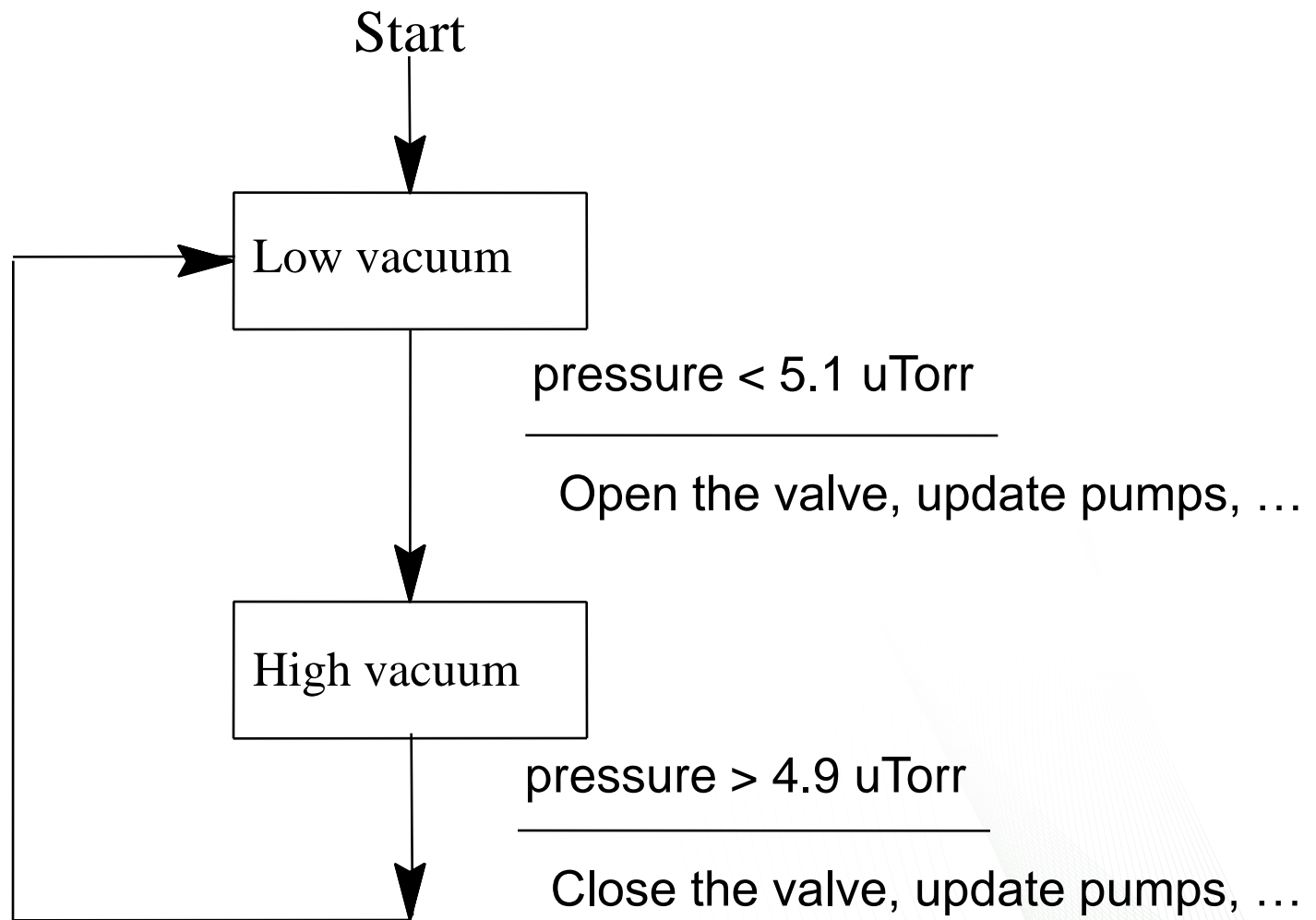


# State Machine 101

- State Machine is in some State
- Events trigger transitions
- Actions are performed on transition



# Example



# Example State Notation Language

```
state low_vacuum
{
```

State

```
    when (pressure <= .00000049)
    {
```

Event

Action

```
        RoughPump = 0;
        pvPut (RoughPump) ;
        CryoPump = 1;
        pvPut (CryoPump) ;
        Valve = 1;
        pvPut (Valve) ;
```

```
    } state high_vacuum
```

Transition

```

}
state high_vacuum
{
    ...
}
```

# How it works

## State Notation Language

```
program sncExample
double v;
assign v to "{user}:aiExample";
monitor v;

ss ss1
{
  state low
  {
    when (v > 5.0)
    {
      printf("sncExample: Changing to high\n");
    } state high
  }
  state high
  {
    when (v <= 5.0)
    {
      printf("sncExample: Changing to low\n");
    } state low
  }
}
```

*"snc"  
Pre-compiler*

## C Code

```
/* Code for state "low" in state set "ss1" */

/* Delay function for state "low" in state set "ss1" */
static void D_ss1_low(SS_ID ssId, struct UserVar *pVar)
{
  # line 15 "../sncExample.stt"
}

/* Event function for state "low" in state set "ss1" */
static long E_ss1_low(SS_ID ssId, struct UserVar *pVar, short *pTransNum, short *pNextState)
{
  # line 15 "../sncExample.stt"
  if ((pVar->v) > 5.0)
  {
    *pNextState = 2;
    *pTransNum = 0;
    return TRUE;
  }
  return FALSE;
}

/* Action function for state "low" in state set "ss1" */
static void A_ss1_low(SS_ID ssId, struct UserVar *pVar, short transNum)
{
  switch(transNum)
  {
    case 0:
    {
      # line 14 "../sncExample.stt"
      printf("sncExample: Changing to high\n");
    }
  }
}
```

*C Compiler*

## Object code

```
00000000 457f 464c 0102 0001 0000 0000 0000 0000
00000020 0001 003e 0001 0000 0000 0000 0000 0000
00000040 0000 0000 0000 0000 1730 0000 0000 0000
00000060 0000 0000 0040 0000 0000 0040 001d 001a
00000100 4855 e589 8948 f87d 8948 f075 c3c9 4855
00001200 e589 8348 10ec 8948 f87d 8948 f075 0ff2
00001400 0510 0000 0000 8b48 f845 00be 0000 4800
00001600 c789 00e8 0000 c900 55c3 8948 48e5 ec83
00002000 4820 7d89 48f8 7589 48f0 5589 48e8 4d89
00002200 48e0 458b bef8 0000 0000 8948 e8c7 0000
00002400 0000 8548 74c0 4819 458b 66e0 00c7 0001
00002600 8b48 e845 c766 0000 b800 0001 0000 05eb
00003000 00b8 0000 c900 55c3 8948 48e5 ec83 4820
00003200 7d89 48f8 7589 89f0 66d0 4589 0fec 45bf
00003400 85ec 75c0 480d 3d8d 0000 0000 00e8 0000
00003600 9000 c3c9 4855 e589 8948 f87d 8948 f075
```

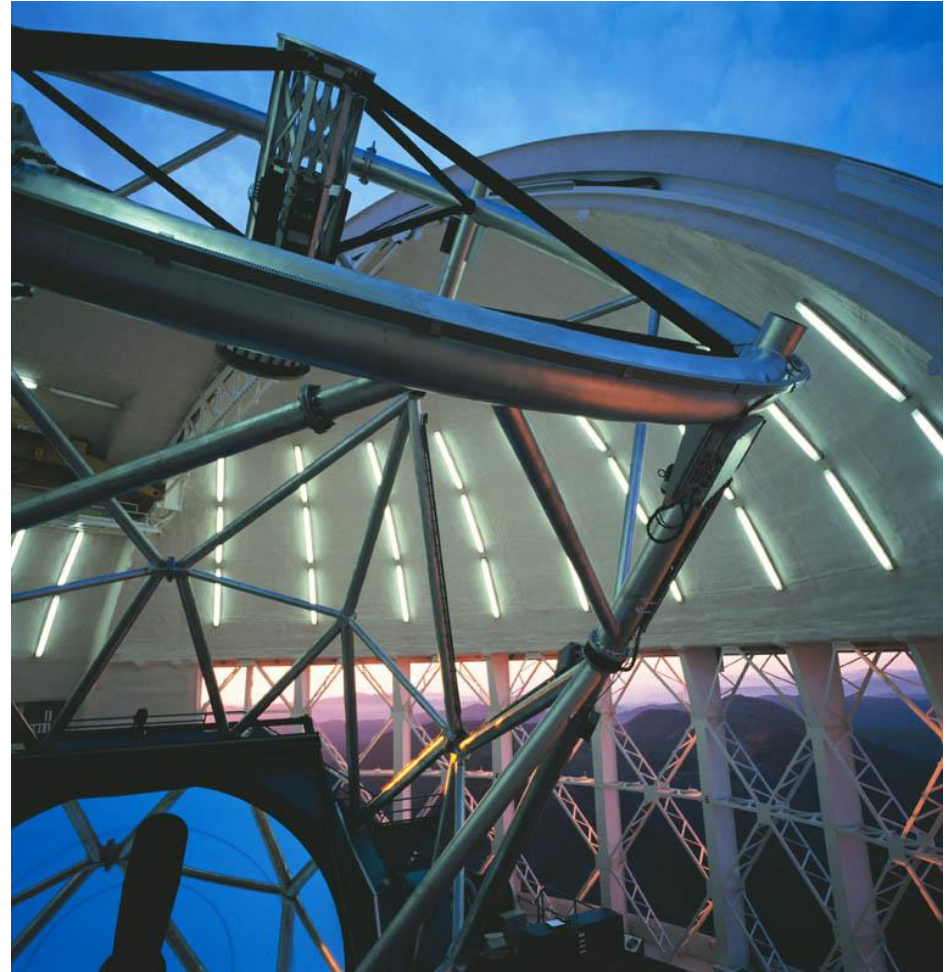
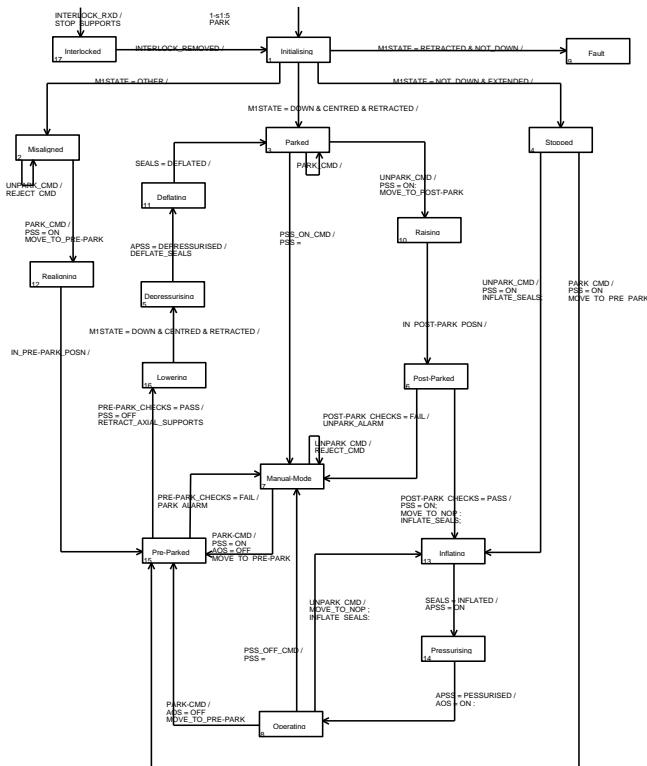
# Advantage

- Compiled code. Fast.
- Can call any C(++) code
  - Use #define to create macros, ...
- Easy connection to Channel Access, Records
  - Compared to custom CA client, device support, ...
- Skeleton for event-driven State Machine
  - Handles threading, event handling, ...



# When to use the sequencer

- For sequencing complex events
- E.g. parking and unparking a telescope mirror



Photograph courtesy of the Gemini Telescopes project



# Disadvantage

- Limited runtime debugging
  - See current state, values of variables, but not details of C code within actions
- Can call any C(++) code
  - and shoot yourself in the foot
- Pre-compiler.  
SNL error
  - SNC creates nonsense C code
  - Totally cryptic C compiler messages
- Risk of writing SNL code
  1. Starts out easy
  2. Evolves
  3. Ends up as a convoluted mess

# Should I use the Sequencer?

## Good Reasons:

- Start-up, shut-down, fault recovery, automated calibration
- Stateful Problem
  - My SNL has 20 states, 30 possible transitions,, and little C code for each transition
- Cannot do this with CALC, BO.HIGH, SEQ, subroutine records

## Bad Reasons:

- PID control, interlocks
- Warning sign:
  - My SNL code has 3 states with 2000 lines of C code
- I don't want to deal with records, I'm more comfortable with C code

# If you really want to use SNL

Good manual:

<http://www-csr.bessy.de/control/SoftDist/sequencer/>

Implement in small steps

- Code a little
- Compile, test
- Code a little more
- Compile, test

# SNL Structure

Program name!

Used in DBD  
And  
to launch the sequence.

```
program SomeName("macro=value")  
/* Comments as in C */  
/* Options */  
/* Variables */  
/* State Sets */
```

# SNL Options

`option +r;`

Make “re-entrant”.

Should be the default.  
Allows running more than one  
copy (with different macros).

`option -c;`

Start right away, do not await  
connections.

Event with “+c”, the default,  
PVs may disconnect..

# Variables

int, short, long, char, float, double

```
double pressure;  
assign pressure to "Tank1Coupler1PressureRB";  
monitor pressure;
```

Map to channel

Update with channel

```
short RoughPump;  
assign RoughPump to "Tank1Coupler1RoughPump";
```

```
string CurrentState;  
assign CurrentState to "{macro}:VacuumState";
```

string == char[40]

Replaced w/ macro's  
value



# Array Variables

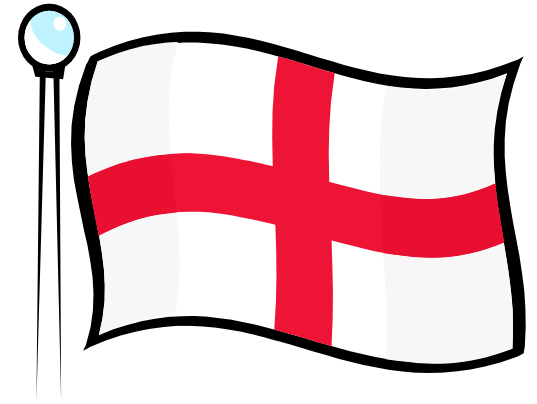
Any but 'string'

```
double pressures[3];  
assign pressures to  
{  
    "Tank1Coupler1PressureRB",  
    "Tank1Coupler2PressureRB",  
    "Tank1Coupler3PressureRB"  
};  
monitor pressures;
```

Map to channel(s!)

```
short waveform[512];  
assign waveform to "SomeWaveformPV";  
monitor waveform ;
```

# Event Flags



- a) Communicate events between state sets
- b) Trigger on Channel Access updates

Declare like this:

```
evflag event_flag_name;
```

Optionally, synchronize with monitored variable

```
sync var_name event_flag_name;
```

# State Sets

First state, name does not matter

```
ss coupler_control
{
    state initial{
        when (pressure > .0000051){
        } state low_vacuum
        when (pressure <= .0000049){
        } state high_vacuum
    }
    state high_vacuum{
        when (pressure > .0000051){
        } state low_vacuum
    }
    state low_vacuum{
        when (pressure <= .0000049){
        } state high_vacuum
        when (delay(600.0)){
        } state fault
    }
    state fault {
    }
}
```

# Events

Variables used in events should be 'monitor'ed!

```
when (pressure > .0000051)
{
  /* Actions ... */
} state low_vacuum
```

```
when (pressure < 0.000051  &&  whatever > 7)
{
} state high_vacuum
```

This is not a wait(10 seconds)!

It means:

After entering the state, if none of the other when(..) events occur within 10 seconds, do this:

```
when (delay(10.0))
{
} state timeout
```

# Events..

Use event Flags:

```
when (efTestAndClear(some_event_flag))  
when (efTest(some_event_flag))  
  
/* Meanwhile, in other state */  
when (pressure < 0.000051  &&  whatever > 7)  
{  
    efSet(some_event_flag);  
} state high_vacuum
```

Check for connections:

```
when (pvConnectCount() < pvChannelCount())  
when (pvConnected(some_variable))
```

# Actions

```
when (pressure > .0000051)
{
    /* Set variable, then write to associated PV */
    RoughPump = 1;
    pvPut(RoughPump);

    /* Can call most other C code */
    printf("Set pump to %d\n", RoughPump);
} state low_vacuum
```

Action statements are almost C code. Above, `RoughPump` is a state machine variable. The SNL is transformed to

```
printf("Set pump to %d\n", pVar->RoughPump);
```

SNC will add the “`pVar->`” to all state machine variables that it recognizes.

Sometimes it will be necessary to

```
%{
/* Escape C code so that it's not transformed */
static void some_method_that_I_like_to_define(double x);
}%
```



## *Walk through the SNL from makeBaseApp -t example*

- ***configure/RELEASE***

*SNCSEQ=/path/to/seq*

- ***Generated Makefile:***

*..\_SRCS += MySource.st*

- ***DBD file entry***

*registrar(SomeNameRegistrar)*

- ***IOC st.cmd***

*seq SomeName, "macro=value"*

# Sequencer Commands

- seq NameOfSequence
  - Start sequence
- seqShow
  - List all sequences with their ID
- seqChan 0x12334
  - Detail of seq.
- seqChanShow 0x12334
  - List variables of seq.
- seqStop 0x12334
  - Stop a sequence

# There is more

- Support for 'entry' and 'exit' blocks
- Assign PV names within code: `pvAssign(..)`
- 'Get Callback', 'Put Callback'
- Checking status & severity of PVs
- 'syncQ' to queue received Channel Access updates

# Summary

- SNL very useful for State-Machine logic
- Read the SNL manual